LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

UCRL-PROC-206797

FUDGE: A Program for Performing Nuclear Data Testing and Sensitivity Studies

B. R. Beck

September 24, 2004

FUDGE: A Program for Performing Nuclear Data Testing and Sensitivity Studies
Santa Fe, NM, United States
September 29, 2004 through October 1, 2004

FUDGE: A Program for Performing Nuclear Data Testing and Sensitivity Studies

Bret R. Beck

Lawrence Livermore National Laboratory

Abstract. We have developed a program called FUDGE that allows one to modify data from LLNL's nuclear database. After modifying data, FUDGE can then be instructed to process the data into the formats used by LLNL's deterministic (ndf) and the Monte Carlo (MCAPM) transport codes. This capability allows users to perform nuclear data sensitivity studies without modification of the transport modeling codes. FUDGE is designed to be user friendly (object-oriented) and fast (the modification and processing typically takes about a minute). It uses Python as a front-end, making it flexible and scriptable. Comparing, plotting and printing of the data are also supported. An overview of FUDGE will be presented as well as examples.

INTRODUCTION

A plethora of computer codes have been written to model nuclear reactors and weapons. These modeling codes require nuclear data (e.g., cross-section and outgoing particle spectra) as input. In general, these codes have no or limited capability to perform sensitivity studies on the nuclear data. That is, the ability to vary cross-section and/or outgoing particle spectra data within reasonable uncertainties in the data and to study how the output varies. In addition, a rewrite of these codes to support sensitivity studies would be unreasonable. A better solution is to provide a way to modify the data before they are inputted to the codes. In this way, only one code needs to be written and maintained for modifying the data, and users only need to learn one interface to modify data.

When writing a code to modify nuclear data, one must understand nuclear data categories and formats. Nuclear data can be divided into three categories: 1) experimental/theoretical, 2) evaluated and 3) processed. Processed data is typically evaluated data converted, via a processing code, into a form required by a modeling code. For example, a deterministic code may require multi-grouped data while a Monte Carlo codes may require equal-probable binned data. Because experimental/theoretical data can require a lot of time and expert knowledge of nuclear physics to evaluate, and because there are various processing codes (like there are various modeling codes), the data modifying code should work with the evaluated data. Of course, after data have been modified, it should be easy to convert it into the required processed data format. Hence, the modifying code must interface with all possible processing codes.

At Lawrence Livermore National Laboratory (LLNL) we have developed a program[1] called FUDGE (For Updating Data and Generating ENDL) that allows users to modify LLNL's evaluated nuclear data and to process the modified data for use in LLNL's modeling codes.

LLNL's evaluated nuclear database, called ENDL[2] (Evaluated Nuclear Data Library), stores distributions (e.g., cross-sections or outgoing particle spectra) as pointwise data. Most evaluated nuclear databases, like ENDF/B-VI, use the ENDF[3] format, which uses a combination of pointwise and parametric form to represent data. At LLNL we have developed codes to convert ENDF formatted data into the pointwise ENDL format and we have converted most ENDF formatted databases into ENDL format.

FUDGE DESIGN REQUIREMENTS

In order for FUDGE to be useful for users, it was determine that it must meet the following criteria:

- FUDGE must be user friendly. It must be easy to retrieve data from an existing evaluation, modify regions of it and process the modification.
- FUDGE must be fast. The time to modify and process must be of order a minute or less.
- FUDGE must run on all computer platforms. This implies that all processing codes must also be portable.
- FUDGE must be scriptable. Users may want to loop over many variations in the data in their sensitivity studies.

To meet most of these requirements, it was decided that the user interface to FUDGE should be written in Python[4]. Python is an object-oriented, interpreted, interactive programming language and comes standard on most UNIX operating systems. FUDGE uses Pythons object-orientation to allow users to easily modify data. For example to increase a cross-section, instantiated by the variable *xsec*, by 10% one simply types in FUDGE (Python)

```
xsec_mod = 1.1 * xsec
```

In this example, the class for cross-section data multiplies the cross-section at each point by 1.1 and stores the result in *xsec_mod*.

HIERARCHICAL FORMAT

FUDGE mimics the hierarchical format of ENDL. At the top of the hierarchy are the available evaluated databases (e.g., ENDL94, ENDL99, ENDF/B-VI). Under each database is a list of incident particles (e.g., neutron, proton or deuteron are possible incident particles in ENDL99). Under each incident particle is a list of targets (isotopes) listed as zaZZZAAA where ZZZ and AAA are the number of protons and nuclei for a target respectively (e.g., za003006 for ^{6}Li and za094239 for ^{239}Pu). The incident particle and target together represent the left-hand-side of a reaction equation (e.g., n + ^{6}Li $\rightarrow$). Under each target is the data for reactions of the incident particle hitting the target.

The top class in FUDGE is the *endlProject* class that represents a database/incident particle hierarchy. When an *endlProject* is instantiated, it initially contains no targets and the user can specify a default database/incident particle to use when reading in a target to be modified. Users specify targets to modify using the *endlProject*'s *readZA* method. The target is retrieved from the *endlProject*'s default database unless another source is specified. Multiple targets from various evaluations can be retrieved. Data from one evaluation can be mixed and matched with other evaluations. Data for the retrieved targets can be modified, saved and processed. For example, the following FUDGE commands read the ^{6}Li target from the neutron incident particle database of the endl99 evaluation, save the modification (modification commands not shown) and create a new deterministic processed data file (ndf1):

```
e = endlProject( database="endl99", yi="neutron" )
Li6 = e.readZA( 3006 )

        (modifications go here)

e.save( )
e.processs( "ndf1", "ndf1.new" )
```

The last command interfaces FUDGE to the processing routines. It uses the first argument ("ndf1" is this example) as a template processed file. FUDGE scans the template file to determine the type of processing to be done (determinist transport in this case) and to determine the parameters needed by the processing code (e.g., group structure and Legendre order information for deterministic transport). The second argument ("ndf1.new") is the name of the file that is to contain the modified data.

DATA CLASSES

FUDGE currently supports 4 types of pointwise data classes. These classes are designated by the number of columns needed to represent the data (1d, 2d, 3d and 4d). For example, cross-sections are 2d pointwise data where column 1 is the energy data and column 2 is the cross-section data. All of these classes support printing (converting the data to a string) and plotting of the data. The classes are called *endl1dmath*, *endl2dmath*, *endl3dmath* and *end4dmath*.

Currently, the 2d data class, called *endl2dmath*, is the most comprehensive. It contains method for adding, subtracting, multiplying and dividing an *endl2dmath* object by a number or another

endl2dmath object (see FUDGE EXAMPLE below). Many other methods exist for the data classes.

FUDGE EXAMPLE

This section presents an example of a FUDGE sessions. In this example, the cross-section for ^{6}Li(n,t)α will be read in from the ENDL99 database and modified by first scaling the cross-section by the value 3.14. Secondly, the cross-section will be scaled by the function show in figure 1.The results are then plotted.

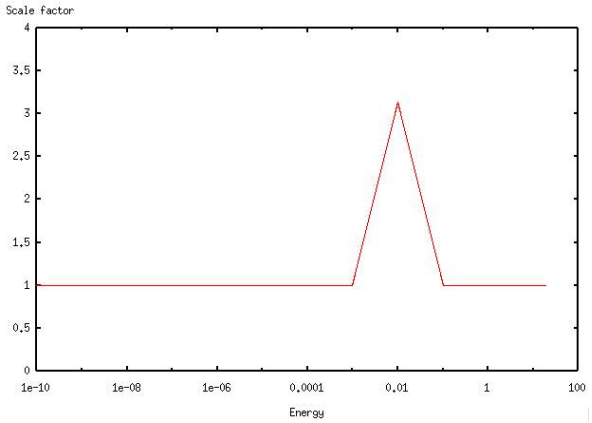

FIGURE 1. Plot of scaleFactor data used in FUDGE example. This plot was make with line 9 of table 1. Figure 2 is the dialog window for this plot.

Table 1 shows the FUDGE commands executed inside Python. Line 1 imports the FUDGE routines. Line 2 instantiates an *endlProject* that uses *endl99/neutron* database as the default when a target is to be retrieved. Line 3 opens the ^{6}Li target from the default database and assigns the variable *Li6* to it. Note that no data has been read in at this point. Line 4 reads in cross-sections data for all reactions. The *I* number specifies the data type (e.g., *I=0* is cross-section data and *I=3* is a type of outgoing spectra data). Line 5 assigns the variable *xsec* to the cross-section for the n + ^{6}Li reaction ^{6}Li (n,t) α. Here, as before, the *I=0* requests cross-section data, while the *C=42* requests data triton as the only outgoing particle (excluding the residual nucleus which in this case is taken to be the outgoing α particle). Line 6 makes a copy of the cross-section with each cross-section value scaled by 3.14. Line 7 creates an *endl2dmath* object consisting of a Python list of (x,y) points (In this example, x is energy and y is a unitless number). A plot of *scaleFactor*'s data is shown in figure 1. Line 8 makes a copy of the cross-section scaled by *scaleFactor*. Note that both *xsec* and *scaleFactor* contain the class *endl2dmath*. This class creates a new *endl2dmath* object whose x values are a union of the x values of *xsec* and *scaleFactor*, and whose y values are the product of *xsec* and *scaleFactor* y values at the corresponding x values (i.e., energy). Linear interpolation is used to fill in missing values.

Line 9 plots *scaleFactor* in an interactive plotting window. A screen capture of the plot is shown in figure 1. This plot can be modified via its dialog window. Figure 2 is a screen capture of its dialog window.

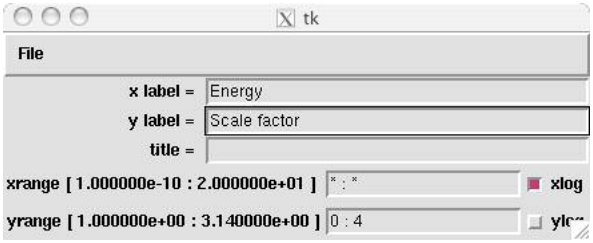

FIGURE 2. This is the dialog window for the plot in figure 1. This dialog allows one to edit the labels and titles, change the x and y ranges. Under the File menu, are options to print, save-as-eps and save-as-ascii.

Line 10 uses the FUDGE function *qmultiPlot* to display several curves on the same plot. The red curve (most of it lies under the blue curve where *scaleFactor* = 1) is the original ^{6}Li(n,t)α cross-section *xsec*. The green curve this *xsec_mod* and the blue curve is *xsec_mod2*. Note that FUDGE makes it easy to modify a region of the *xsec* data, allowing for sensitivity studies on a region of the data.

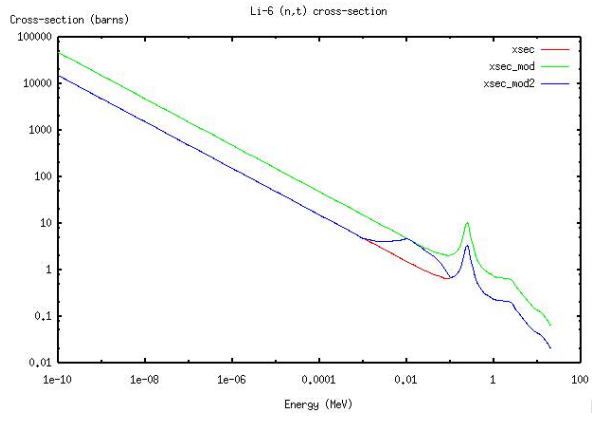

FIGURE 3. Plot of ^{6}Li(n,t)α cross-section. The red curve is the original ENDL99 data. The green curve is the original data scaled by 3.14. The blue curve is the original

data scale by the function shown in figure 1. This plot was made with line 10 of table 1.

TABLE 1. Example FUDGE commands

#	FUDGE command	Comment
1	>>> from fudge import *	# Get the FUDGE routines
2	>>> e = endlProject(database = “endl99”, yi = “neutron”)	# Create a new project
3	>>> Li6 = e.readZA(3006)	# Get the Li-6 target
4	>>> Li6.read(I = 0)	# Read in all cross-sections
5	>>> xsec = Li6.findData(C = 42, I = 0)	# Get (n,t) cross-section
6	>>> xsec_mod = 3.14 * xsec	# Scale xsec by 3.14
7	>>> scaleFactor = endl2dmath([[1e-10, 1], [1e-3, 1], [0.01, 3.14], \ … [1e-1, 1], [20., 1]])	# Define a vector
8	>>> xsec_mod2 = scaleFactor * xsec	# Scale xsec by scaleFactor
9	>>> scaleFactor.plot( )	# Plot scaleFactor
10	>>> qmultiPlot([xsec, xsec_mod, xsec_mod2], legends = ["xsec", "xsec_mod", \ … “xsec_mod2"], xylog = 3, title="Li-6 (n,t) cross-section")	# Plot xsec, xsec_mod and # xsec_mod2 in one plot

IMPROVEMENTS

The current version of FUDGE is written using Python lists to store data. These can be very inefficient when searching for a value in the list. Python lists also require a lot of memory. FUDGE is being modified to store the data using Numerical Python[5] arrays. The ENDL format does not allow for uncertainty values. A redesign of the ENDL format is in progress and FUDGE needs to be updated to handle uncertainty values. FUDGE needs to have more 3d and 4d math functionality. Finally, add a Graphical User Interface (GUI) to FUDGE.

ACKNOWLEDGMENTS

This work was performed under the auspices of the U.S. Department of Energy by Univ. of California, Lawrence Livermore National Laboratory under contract No. W-7405-ENG-48.

REFERENCES

1. Here “program” is used to designate the modifying code and other required codes (e.g., the processing codes).

2. Howerton, R. J., MacGregor, M. H., “The LLL Evaluated Nuclear Data Library (ENDL): Description of Individual Evaluations for Z=0,98”, Lawrence Livermore National Laboratory Report UCRL-50400 Vol. 15 (1978)

3. US Evaluated Nuclear Data Library, ENDF/B-VI, Release 8 – Produced by the Cross Section Evaluation Working Group, released in October 2001, available at www.nndc.bnl.gov/csewg/.

4. G. van Rossum and F.L. Drake (eds), Python Reference Manual, PythonLabs, Virginia, USA, 2001. Available at http://www.python.org.

5. D. Ascher, P.F. Dubois, K. Hinsen, J. Hugunin and T. Oliphant, Numerical Python, Lawrence Livermore National Laboratory, Livermore, California, USA, 2001. Available at http://www.pfdubois.com/numpy/.